Ejercicios de árboles

Instrucciones Se incluye un archivo EvaluarArbol.hpp que tiene una clase EvaluarArbol dentro del namespace aed. Esta clase tiene 5 métodos que se pueden utilizar para autoevaluar los ejercicios. Los nombres de los métodos son evaluar1, evaluar2, ..., evaluar5 para los ejercicios 1, 2, ..., 5 respectivamente. Cada una recibe un parámetro, correspondiente a la función que resuelve el ejercicio. Dicha función se indica en cada ejercicio.

Un ejemplo de uso para evaluar todos los ejercicios es:

```
#include "EvaluarArbol.hpp"
using namespace aed;
int main(int argc, char *argv[])
{
        EvaluarArbol ev;
        ev.evaluar1(tree_equal);    // Ejercicio 1
        ev.evaluar2(erasing_equal); // Ejercicio 2
        ev.evaluar3(colapsar_nodo); // Ejercicio 3
        ev.evaluar4(unir_suma);     // Ejercicio 4
        ev.evaluar5(suma_nivel);    // Ejercicio 5

        return 0;
}
```

En principio puedes comentar los ejercicios que no quieras evaluar o no hayas resuelto aún.

Para asegurar que todo compile y se ejecute correctamente, si se utiliza zinjai, se recomienda crear un proyecto, y en el incluir EvaluarArbol.hpp, util_tree.h, util_tree.cpp, util.h, util.cpp, tree.h y tu archivo main.cpp donde tengas el main. Además, todos estos archivos, tenerlos en la misma carpeta.

Ejercicio 1 Escribir una función

```
bool tree_equal(tree<int> &T1, tree<int> &T2)
```

que devuelve true si T1 y T2 son iguales. Que sean iguales significa que tengan la misma forma y los mismos valores en cada nodo.

Ejemplos:
`T1 = (1 2 (3 4 5))`
`T2 = (1 2 (3 4 5))`
Son iguales

`T1 = (1 (3 4 5) 2)`
`T2 = (1 2 (3 4 5))`
No son iguales

Ejercicio 2 Escribir una función

```
bool erasing_equal(tree<int> &T1, tree<int> &T2)'
```

que devuelva true si es posible obtener T1 aplicando erases sobre T2. Tener en cuenta que al borrar un nodo, se borra todo el subárbol.

Ejemplos:
`T1 = (1 3 (4 5 6))`
`T2 = (1 (2 11 12) 3 (4 5 (6 9 10) 7))`
Devuelve true.
Si de T2 borramos el los subárboles (2 11 12), 7, 9, 10, obtenemos T1.

`T1 = (1 3 4)`
`T2 = (1 (2 11 12) 3 (4 5 (6 9 10) 7))`
Devuelve true.
De T2 se borra (2 11 12), 5, (6 9 10) y 7.

`T1 = (1 4 3)`
`T2 = (1 (2 11 12) 3 (4 5 (6 9 10) 7))`
Devuelve false.
En T1 el nodo con valor 3 está a la derecha del 4, es imposible lograr esto borrando elementos de T2.

Ejercicio 3 Escribir una función

```
void colapsar_nodo(tree<int>& T, int n)
```

donde se debe eliminar el nodo que tiene de valor n, pero no todo el subárbol. Sino, subir de nivel sus hijos. El árbol no tiene valores repetidos, y n no representa a la raíz.

Ejemplos:
`T = (1 (2 3 4))`
`n = 2`
T debe quedar: (1 3 4)
Los hijos de 2 se suben de nivel y el 2 se borra.

`T = (1 (2 (3 4) 5) 6)`
`n = 2`
T debe quedar: `(1 (3 4) 5 6)`
Los subarboles hijos de 2 suben de nivel, y el 2 se borra.

Ejercicio 4 Escribir una función

```
void unir_suma (tree<int> &T1, tree<int> &T2)
```

donde T2 es un árbol vacío. Todos los nodos de T1 que tienen un solo hijo deben unirse con su hijo y en dicho nodo debe quedar como valor la suma de su valor mas la de su hijo. El árbol resultante se debe guardar en T2. Notar que el árbol resultante no tendrá ningún nodo con sólamente un hijo.

Ejemplos:
`T1 = (1 1)`
T2 debe quedar = `1` (un solo nodo ra´iz = 1)

`T1 = (1 (4 6) 2)` T2 debe quedar = `(1 10 2)`, notar que el nodo 4 tiene un solo hijo, el 6, entonces se unen y queda la suma, 10 = 4 + 6

`T1 = (1 2 (1 (1 1)))`
T2 debe quedar = `(1 2 3)`
Ahora hay dos nodos que tienen un solo hijo, se deben unir todos, entonces 3 = 1 + 1 + 1

`T1 = (1 (2 (2 (3 1 5))) (4 6 (7 8)))`
T2 debe quedar = `(1 (7 1 5) (4 6 15))`

Ejercicio 5 Escribir una función

```
void suma_nivel (tree<int> &T, list<int> &L)
```

que cargue en L la suma de todos los nodos de cada nivel de T, ordenados del nivel menor al mayor, siendo el primer nivel el correspondiente a la raíz.

Ejemplos:
`T = (1 (2 (3 (4))))`
L debe quedar = `[1 2 3 4]`
Cada nivel tiene un solo nodo

`T = (1 1 1)` L debe quedar = `[1 2]`
El último nivel tiene dos nodos con valor 1, la suma es 2.

`T = (3 1 (1 4 4) (2 (2 1) 5))`
L debe quedar = `[3 4 15 1]`